

SIGNAC: A SIMPLE DATA MANAGEMENT FRAMEWORK

CARL S. ADORF

*Department of Chemical Engineering, University of Michigan, Ann Arbor, MI
48109*

PAUL M. DODD

*Department of Chemical Engineering, University of Michigan, Ann Arbor, MI
48109*

SHARON C. GLOTZER

*Department of Chemical Engineering, University of Michigan, Ann Arbor, MI
48109*

*Department of Materials Science and Engineering, University of Michigan, Ann
Arbor, MI 48109*

Biointerfaces Institute, University of Michigan, Ann Arbor, MI 48109

ABSTRACT. Researchers in the field of computational physics, chemistry, and materials science are regularly posed with the challenge of managing large and heterogeneous data spaces. The amount of data increases in lockstep with computational efficiency multiplied by the amount of available computational resources, which shifts the bottleneck within the scientific process from data acquisition to data post-processing and analysis. We present a framework designed to aid in the integration of various specialized data formats, tools and workflows. The **signac** framework provides all basic components required to create a well-defined and thus collectively accessible data space, simplifying data access and modification through a homogeneous data interface, largely agnostic of the data source, i.e., computation or experiment. The framework's data model is designed not to require absolute commitment to the presented implementation, simplifying adaption into existing data sets and workflows. This approach not only increases the efficiency with which scientific results can be produced, but also significantly lowers barriers for collaborations requiring shared data access.

E-mail addresses: csadorf@umich.edu, pdodd@umich.edu, sglotzer@umich.edu.

1. INTRODUCTION

Better software[21, 4, 6, 5, 2] and increased amounts of computational resources available to researchers in the field of scientific computation[22, 25] have led to a significant increase in the amount of data required to be stored and analyzed[14]. While high-dimensional and heterogeneous data spaces generally increase chances to reveal broader relationships[10, 8], they also significantly increase the complexity of data management. This complexity shifts the bottleneck within the computational process from the acquisition of data to the post-processing of data and also raises barriers for collaborations relying on shared data.



FIGURE 1. The Pointillist style was invented by Paul Signac (1863-1935) and Georges Seurat (1859-1891) and describes paintings, which are composed of many individual dots painted in a single color, that in union form a natural image. This style serves as a metaphor for **signac**'s data model, where both the individual data points but also their position within the larger data space compose an interpretable result. The painting underlying this artistic illustration *Cassis, Cap Lombard* was created by Paul Signac in 1889 and is owned by the Gemeentemuseum in Den Haag.

Here we present the **signac** framework, which provides functionality for operating on data spaces common to a variety of computational workflows, such as reliably and efficiently associating data and metadata, iterating over data spaces, selecting data subspaces, reorganizing data, indexing data, and exporting data into databases. These integrated functionalities allow a researcher who uses **signac** to concentrate on the implementation and execution of a specific computational workflow rather than needing to repeatedly implement these basic functionalities for each new project. The presented software is focused on providing simple solutions to aforementioned problems with a very low entry barrier and does not present

a complete solution to all data management related challenges. In particular, we assume that any infrastructure related issues, such as the setup of and access to a distributed file system are better addressed and solved by systems such as the Integrated Rule-Oriented Data System (iRODS) [23] or GLOBUS [11] both of which are targeted at a different scope than **signac**.

The framework is named for painter Paul Signac, who along with Georges Seurat, developed the Pointillism style; art that is only visually accessible by viewing it from a certain distance; see fig. 1 for an example of the described technique. Similarly, the **signac** framework is aimed to aid the generation and management of fine-grained heterogeneous data spaces, which analyzed in unison reveal a bigger picture.

Another major advantage of using a data management framework such as **signac** is the explicit documentation of how data is organized. In this sense **signac** acts like a database providing a clearly defined interface to data and metadata, making them searchable and thus accessible. This allows anyone to access the data, not only those actually using the framework to implement their own workflows. There is strong evidence that well-maintained public databases, such as the Protein Data Bank (PDB)[7], the Cambridge Structural Database (CSD)[3, 13], The Materials Project[14] or ImageNet[9] have a significant positive impact on their respective fields. Promoting an open data culture among researchers within one or across multiple organizations will likely result in similar positive synergistic effects.

2. PROBLEMS AND BACKGROUND

2.1. Data and Metadata Management. For the implementation of this framework we assume that a computational investigation operates on a discrete data space within a high-dimensional parameter space, as visualized in fig. 2. The parameters associated with the data are part of the metadata, which is essential for interpretability and reproducibility.

A simple and robust solution to associate data files with metadata is to store files with a well-defined path within a file system. The file system is usually the primary way to store data in the context of high-performance computing (HPC), because it is the fastest and on adequately backed-up systems, the most reliable way to store data. The file system abstracts complexity of the underlying system and network infrastructure, however there are limitations: 1. Most modern file systems limit the choice and number of characters to be used for a single file path, thus limiting the amount of metadata to be stored in the file path in many cases to 255 bytes. 2. Many file systems are inefficient when it comes to large flat hierarchies, which means many files stored within only a few directories. 3. File paths are largely inflexible. As a result, changes to the metadata hierarchy, for example, are difficult to handle. 4. Search operations on file paths are largely inefficient. It is therefore advantageous to reduce the amount of metadata stored as part of the file path to a minimum and instead store the full set of metadata in a different way.

An alternative approach is to store data in a database via a database management system (DBMS), such as MongoDB [17] or MySQL [18], which are designed to handle vast amounts of data and metadata. Such an approach reliably associates data and parameters, allows execution of complex query operations, and is generally more flexible. However, database systems are usually less efficient in handling large chunks of (binary) data, which is why it is common practice to store data files

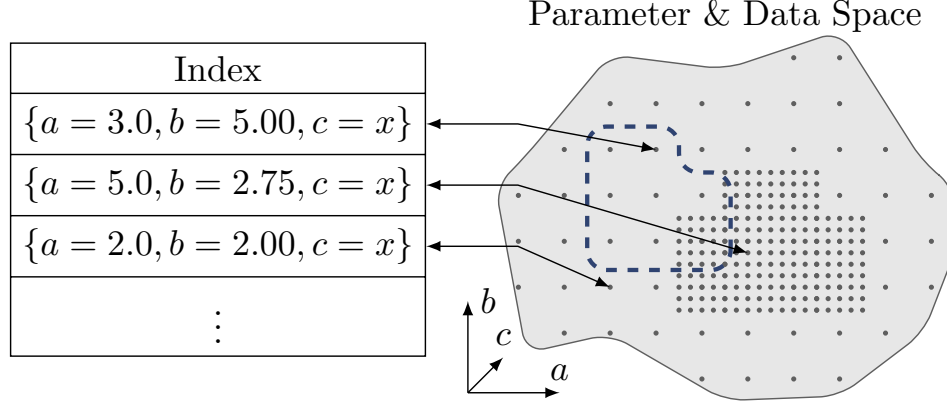


FIGURE 2. The parameter space is a high-dimensional space spanned by all possible parameter ranges affecting our data. Each dot within the parameter space represents a data point, which is uniquely addressable via a unique set of parameters. The union of all data points forms the data space, while a parameter set represents a coordinate within that space. Only the combination of coordinate and associated data form an entity, which is accessible to interpretation. **signac** ensures the integrity of the data space by tightly coupling each operation on the data space to the associated unique set of parameters. It also simplifies the addressability of a parameter subspace, such as the one visualized by the blue boundary, which can be expressed as the union of two parameter ranges: $\{1.5 < a < 4.75 \quad 2.5 < b < 4.5 \quad c = x\} \cup \{1.5 < a < 3.5 \quad 4.5 \leq b < 5.5 \quad c = x\}$.

in a well-defined manner in the filesystem and only link to these files, e.g. via the file path within the database.

In the realm of HPC, database systems may pose a significant bottleneck for data processing and are not necessarily straightforward to deploy. The **signac** framework combines the best of both worlds by supporting the user in storing and processing data on the file system, while reliably and transparently keeping track of all associated metadata. Secondly, **signac** provides functions to create an index of all data stored in this way, which can be exported into a database. Finally, **signac** enables the user to transparently access data files via the metadata stored in the index. This combined approach enables the user to avoid all performance and storage bottlenecks associated with the use of databases for data handling during HPC data processing, while still taking full advantage of advanced metadata handling capabilities of modern database implementations (fig. 3).

By decoupling data processing and index creation into two distinct steps, it is also more easily possible to create indexes of already existing data that is stored in the file system with a different schema. This approach is enormously powerful in providing a single homogeneous data interface for new and existing data, which may be made accessible to individual researchers within and across organizations or even publicly.

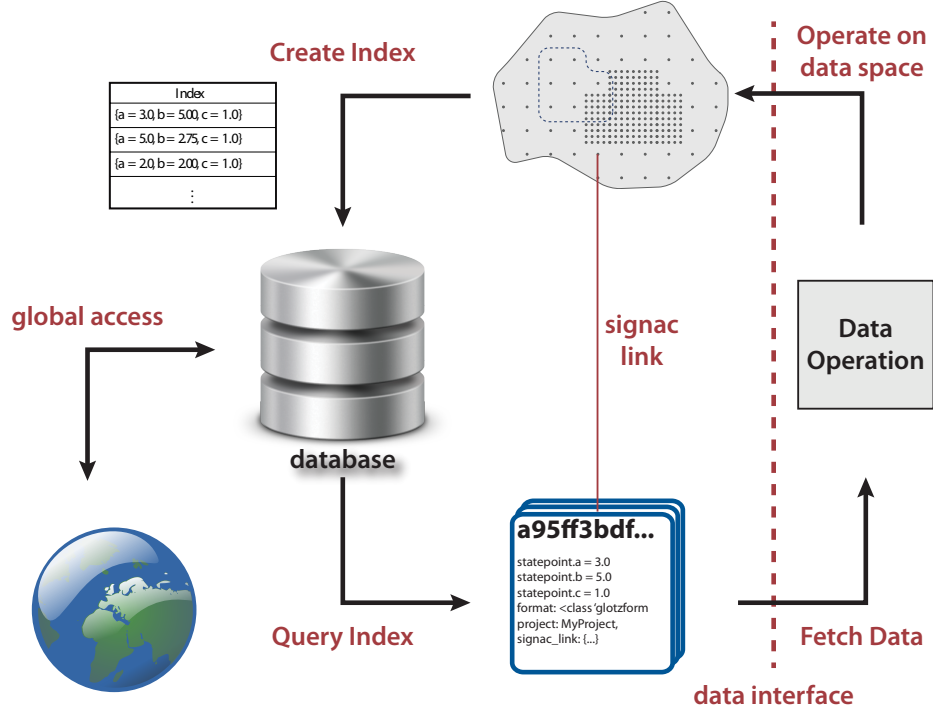


FIGURE 3. The **signac** framework supports the user in operating on and indexing a well-defined data space within a file system. This index contains all information required to reliably link data and metadata. This combined approach allows the user to take full advantage of a database’s capability for simple or complex query operations, without a performance penalty that may be associated with an approach that relies on a database system during all stages of the process.

2.2. Workflow model. Recognizing that most researchers and organizations working computationally use a mix of post-processing and workflow-assistance tools, **signac** is designed to *not* act as a black box. In contrast to DCMS[15] and the AiiDA infrastructure[20], **signac** explicitly avoids the requirement to express a computational workflow through one general infrastructure language. Instead we ensure that the framework is structured like an onion with multiple layers, where the data model represents the core and the data interface and the workflow routines are layered around it. Dependencies between individual components point only outwards. This means, e.g., that entities at the center, such as the data model, do not depend on higher-level components. Such a structure follows the *clean architecture* principle[16] and among other things means that the system is more flexible in adapting to architectural changes. A practical implication is that it is easily possible to implement routines to read from or write to a valid **signac workspace** independent from a more complete implementation of the framework. This increases the flexibility in supplementing existing workflows, but also means that data remains accessible without **signac**, which is crucial for reliable and sustainable long-term

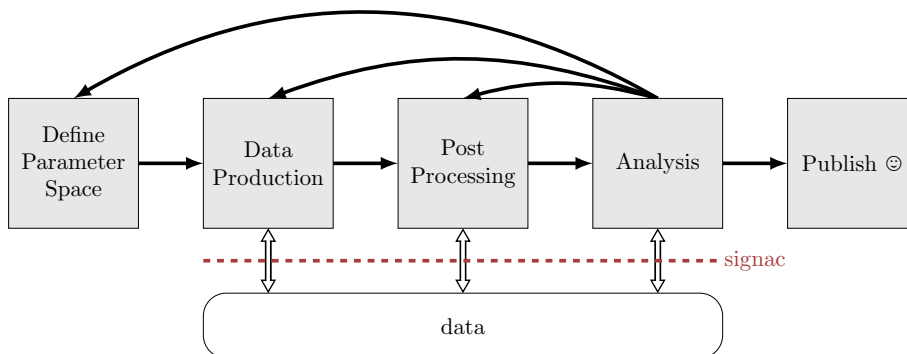


FIGURE 4. The sample workflow captures core processes that are common to many computational (and even experimental) workflows and guides the development of the framework. Rather than owning the data, **signac** is designed to manage the flux at the interface of the data space and these processes. This means accessing the data via **signac**’s Application Program Interface (API) ensures the integrity of the data space, but is not required for data access. This guarantees that the data remains accessible even when a **signac** implementation is not available.

data storage. In other words, the **signac** framework provides transparent solutions to recurring problems common to most computational workflows.

Three core processes that are common to almost all computational workflows are data acquisition, data post-processing and data analysis (fig. 4). These processes guide the design and implementation of the framework. We define data acquisition as any process that generates or procures raw data, such as simulations or experiments. Post-processing is any operation that refines and prepares raw data for direct analysis, specifically those that are associated with significant computational or storage costs. Analysis operations transform all data into a form that is interpretable by a researcher, e.g., in the form of tables, plots, fitted models and statistical properties. While steps may be added or removed in this example workflow, the role **signac** plays does not change and we will use this example without loss of generality. Data acquisition and post-processing steps usually operate in a higher cost regime and at longer time scales, while the analysis process is ideally conducted almost instantaneously. An example for data acquisition may be the execution of compute-intensive density functional theory calculations or molecular dynamics simulations or the collection of data from experiments. In post-processing, the acquired data maybe further non-trivially refined and computationally evaluated. In a final step a researcher analyzes the data in real-time in order to compile the data into results and draw conclusions. A clear distinction between these individual processes may not always be possible or beneficial, however the model serves as a general guideline in the design of the framework. **signac** aids these steps in any workflow by managing the data, allowing for easy data access.

Component	Function
project data management	Simplifies the management of and access to complex and heterogeneous data spaces.
indexing	Creates data indexes for advanced post-processing and analysis routines.
database integration	Exports indexes and data to private and/or public databases.
graphical user interface	Basic interface for configuration and data structure visualization.

TABLE 1. The **signac** framework is divided into four core components, each of them providing specific functionality for different stages of a computational workflow.

3. SOFTWARE FRAMEWORK AND IMPLEMENTATION

3.1. Software Architecture. The **signac** framework is divided into four core components: project data management, indexing, database integration and the graphical user interface (tab. 1). The first two components are designed to be used in a high-performance computing environment, which is why any requirements besides the python interpreter for these functionalities are avoided.

The **signac** framework is implemented in python tested for versions 2.7.x and 3.x. The package has very high test coverage and is documented using the Sphinx documentation tool[12]. Additional functionalities require additional libraries, such as **pymongo** for MongoDB database integration and **PySide** for the graphical user interface (GUI).

Metadata is encoded in the open standard JavaScript Object Notification (JSON) format, for which parsers and writers are available in most programming languages and computing environments and which is largely human readable. Relying on a simple, open format is important to ensure that data remains accessible even without **signac**. Furthermore many NoSQL database systems internally rely on the JSON format, allowing an effortless integration of these systems.

3.2. Software Components. The following sections discuss the four individual core components in more detail.

3.2.1. Project Data Management. All processes identified in the workflow model (section 2.2) — data acquisition, post-processing, and analysis — require consistent and simple data access. The **signac** framework addresses this requirement by providing a transparent but homogeneous API for all data related processes related to one project, as depicted in fig. 4. Although **signac** is implemented in python, the API is also accessible on the command line, not limiting it to python-oriented workflows.

The interface model is based on the assumption that computational investigations are divided into projects, where each project is confined by roughly similarly structured data. Data associated with a project is allocated to a *workspace*, see

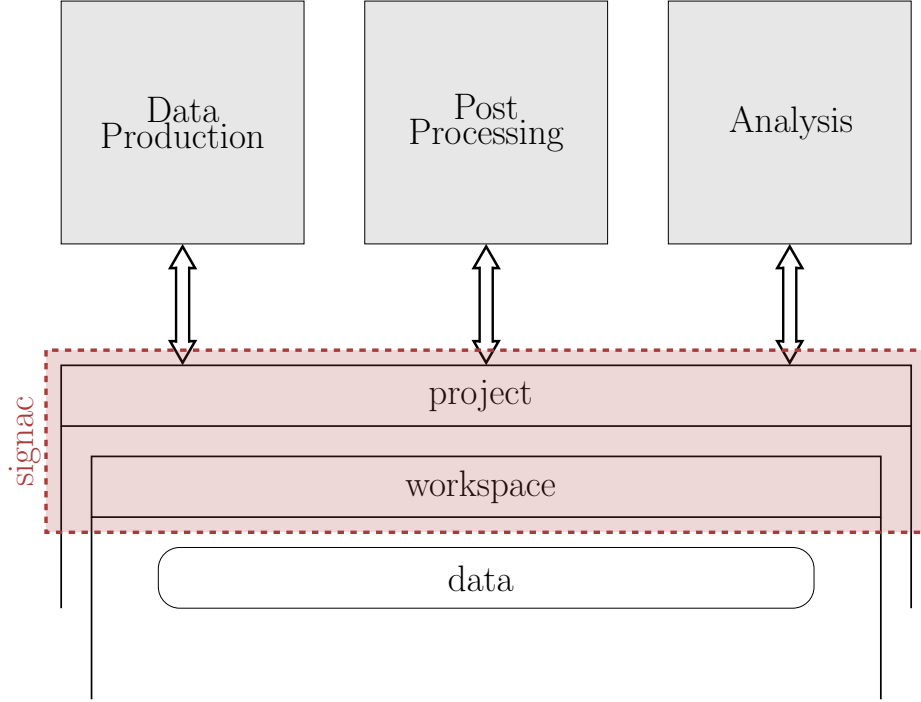


FIGURE 5. The interface to the data space illustrated as red dashed line in fig. 4 consists of two layers. The first layer is a project layer, which gives access to the data interface. The second layer is a workspace layer, which represents the allocation of data to a physical storage. These two layers do not form a one-to-one relationship, a project may interface multiple workspaces and a workspace may be interfaced by multiple projects. That means a workspace is self-sufficient and project agnostic. Providing one interface for all processes requiring access to data for reading or manipulation makes routines implementing those processes robust against changes in the parameter or data space.

fig. 5. Any project-related process operates on data within the *workspace* by reading, adding, removing or modifying data. The *workspace* is project agnostic, i.e. a project may interface many *workspaces*, and a *workspace* may be interfaced by many projects. The core functionalities provided by the project API include, but are not limited to, the exposure of *workspace* paths; routines to create, search and iterate *workspace* data; and the creation of additional *views* of the *workspace*.

The main challenge of reliable long-term storage of data in a file system or a database is to ensure the proper association of data and metadata. This association needs to be guaranteed even if an implementation of **signac** were not available to a user accessing such data. All data needs to be uniquely addressable within the data space from the associated metadata and *vice versa*.

For this purpose we define an identifier, called *job id*, which is a clearly defined hash value calculated from the metadata. This hash value is then used as a unique and reliable address of data in all contexts. The *job id* represents a better address of data than a plain text file name or a database table, because it is short, non-ambiguous, and thus indexable.

Using a hash value for addressing is convenient and allows fast random-access. The **signac** framework stores a full copy of all metadata in plain text within the workspace and additionally supports the creation of hash tables for the reverse look-up. This storage model is completely transparent and data access does not depend on the availability of **signac**.

Because all data are stored in *workspace* paths obfuscated by the *job id*, the *workspace* is harder to navigate. The framework allows one to create *views*, which are human readable symbolic links to the actual paths. *Views* make it possible to browse through data spaces on the file system directly. They are more flexible than hard-coded paths, because they can be limited to certain data subspaces. Furthermore, parameters that are constant over the view space need not be part of the view path. A view path is as long as needed, but as compact as possible; see section 4.3 for an example.

3.2.2. Indexing. Based on the *job id* it is simple to create a *data index*, which contains all information about the data structure of a specific data set in one place. This simplifies the mapping between different, possibly heterogeneous, storage devices, such as a file system and a database.

The index is generated with one or more *crawlers*. For our purposes a *crawler* is any function that generates a series of JSON documents. In general, a *crawler* function is not bound to any storage device and is thus highly versatile. The **signac** framework provides templates for *crawlers* that crawl through file systems.

Generally, the index needs to contain the metadata associated with the data and all information required to allow access to the data. In the specific case of a file system index, this is the metadata and information about file locations and formats.

3.2.3. Database Integration. The **signac** framework is designed to facilitate the integration of databases into the workflow. It provides routines for the creation of indexes, as discussed in section 3.2.2, and furthermore provides routines for the export of indexes.

The current implementation supports the MongoDB NoSQL database, but in principle any database system that provides a python driver could be integrated in the future. We chose MongoDB because its internal data structure is already based on the JSON format and because we consider the NoSQL approach more flexible and intuitive to researchers, who are not usually required to define a table schema for their data structures.

3.2.4. Graphical user interface. The graphical user interface (GUI) enables the graphical inspection of data structures, which is useful for data survey and the development of database query routines. As the GUI is part of the framework, it shares configuration and query language with **signac**'s other components.

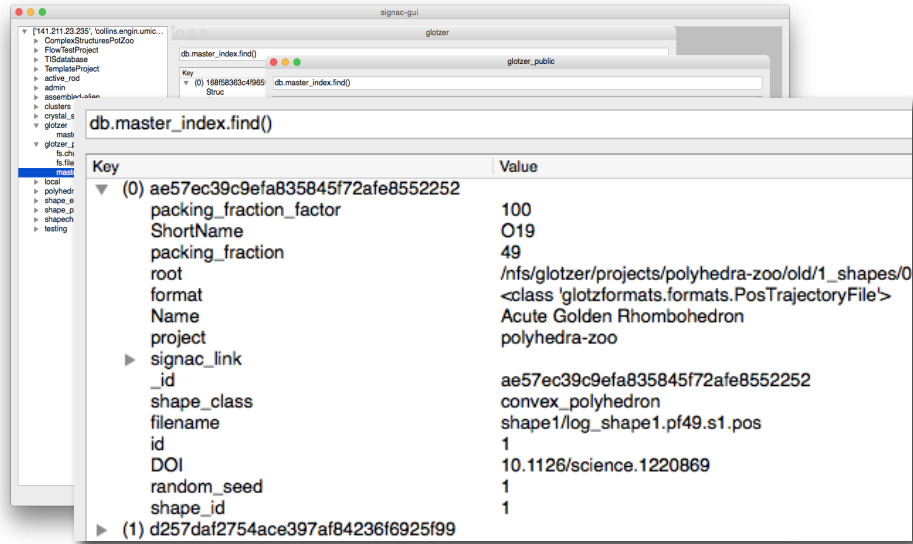


FIGURE 6. The graphical user interface (GUI) provides basic functionality for configuration and visualization of data in JSON format.

The GUI is designed to provide minimal functionality for the above described purpose. It does not support data manipulation or the execution of aggregation operations, because there are specialized open-source tools available for that purpose (e.g. *MongoChef*[1] or *Robomongo*[19]).

4. EXAMPLE APPLICATIONS

All listings are written in the python programming language unless otherwise stated.

4.1. Data access – minimal example. After the configuration of a project, the basic data management API is accessed through a *project handle*:

```
import signac
project = signac.get_project()
```

Each operation on project data requires a *job handle* for a specific set of parameters. In this example the parameter set contains only a single key-value pair: $\{a : 0\}$. The *job handle* then provides a unique reference point for data associated with this particular set of parameters, such as the *job id* or the physical storage path of the associated data.

```
# We obtain a job handle to manipulate data
# associated with a unique set of parameters,
job = project.open_job({'a': 0})

# which ensures the proper
# association of job id
print(job.get_id())
```

```
9bfd29df07674bc4aa960cf661b5acd2
```

```
# and statepoint:
print(job.statepoint())
{'a': 0}
```

4.2. Creating an index. Creating an index on project data is simplified by specializing one of the **signac** *crawler* implementations. As described in section 3.2.2, a *crawler* crawls through the data space and collects information about the data, namely the metadata and how to access the data. The index itself is a collection of JSON documents, which can then be exported into a NoSQL database.

The following example demonstrates how to index the project data space taking advantage of the standardized **signac** storage structure:

```
formats = {'.*\\.txt': TextFile}
for doc in project.index(formats):
    print(doc)
```

The regular expression is used to declare that all files with file names ending in `.txt`, are to be indexed and are of *TextFile* format. Any metadata associated with this file is automatically retrieved from **signac**'s data model and is part of the index.

4.3. Generating a phase diagram with molecular dynamics simulations.

This slightly more elaborate, but still generic, example demonstrates the use of **signac** to manage an ensemble of molecular dynamics simulations performed at different thermodynamic state points. For simulations, the metadata associated with data generally consists of all parameters required to execute the simulation that are essential for interpretation and reproducibility. This example demonstrates how to simulate a system of point particles interacting with a Lennard-Jones (LJ) interaction potential at different state points (i.e. at different pressures) to generate a phase diagram [24].

The first step is to clearly define the state point as a function of the variable(s) of interest, in this case the pressure. Any required input parameter that would change the simulation output is part of this definition.

```
def get_statepoint(pressure):
    return {
        'N': 1000,
        'diameter': 1.0,
        'temperature': 1.0,
        'pressure': pressure,
        'tau': 0.5,
        'tauP': 1.2,
        'sigma': 1.0,
        'epsilon': 1.0,
        'r_cut': 2.5,
        'dt': 0.005,
        'phi_p_init': 0.05,
        'random_seed': 42,
    }
```

Next, the simulations are executed sequentially by iterating over the variable of interest. Using the `signac` API ensures a consistent association of the state point with the output.

```
project = signac.get_project()

# Iterate over multiple pressures
for pressure in [0.01, 0.1, 1.0, 10.0]:

    # Generate full statepoint
    statepoint = get_statepoint(pressure)

    # Obtain a job handle for this statepoint
    with project.open_job(statepoint) as job:

        # Setup and run the simulation
        setup(job)
        run(10000)
```

To inspect the data directly on the file system, it is useful to create a *view* of the *workspace*. Creating a *view* generates a human readable directory hierarchy containing *symbolic links* to the actual *workspace* directories. The majority of variables in this example do not vary across the whole data space. These variables are automatically omitted during the creation of the *view* hierarchy to maximize its compactness. A *view* in this example would generate a one-dimensional hierarchy of links that point to data directories associated with the different pressures:

```
$ ls view/pressure
0.01 0.1 1.0 10.0
```

One way to create the phase diagram is to simply iterate over the data space supported by `signac`'s API:

```
ps = list()
Vs = list()

# Iterate over the workspace
for job in project.find_jobs():

    # Retrieve the associated pressure
    p = job.statepoint()['pressure']
    ps.append(p)

    # Retrieve the calculated volume
    V = get_volume(job)
    Vs.append(V)

plot(ps, Vs)
```

This simple iteration is possible because the data space for this example is only one-dimensional.

However, the introduction of just one more dimension would immediately trigger the requirement for more sophisticated sorting and aggregation techniques. This

is why **signac** encourages users to build an index for this data, which then allows one to perform operations such as filtering or grouping.

5. CONCLUSIONS

The development of **signac** is motivated by the increased need for the management of heterogeneous and complex data spaces. We present **signac** as a software framework to manage heterogeneous and complex data spaces that arise in computational science. We find that researchers in computational fields are repeatedly required to solve very similar problems posed by recurring tasks such as the reliable codification of metadata, post-processing in heterogeneous data spaces and the description of data structures. The overall data processing efficiency is improved by combining the strengths of different data carriers, such as file systems and databases. The software presented here poses a non-intrusive solution to many data management and workflow challenges in environments scaling from desktop applications to high-performance cluster computing.

The **signac** framework relies on the robust, open standard JSON format for metadata storage and employs a data model that is simple enough that it can be easily accessed without the presented implementation. This is important for sustainable long-term storage and lowering inhibitions for usage. Any framework that requires full commitment to a specific implementation involves the risk of losing access to data stored in the specific schema.

The homogeneous interface promotes transferability of workflows. The project data model represents a basic agreement on how data is handled. In the ideal case, a workflow developed for a specific data set requires only minor adjustments in the query formulation to be applicable to a completely different data set. This flexibility is critical for rapid workflow development and promotes an open data environment.

ACKNOWLEDGMENTS

We would like to thank J.A. Anderson for fruitful discussion, feedback and support, M.E. Irrgang for helpful discussion and feedback, P. Damasceno for early discussion of the concept and support, and B. Swerdlow for early feedback and coming up with the name. We would also like to thank all early adopters that provided feedback and thus helped in guiding the development process. Project conceptualization and implementation was supported by the National Science Foundation, Division of Materials Research Award # DMR 1409620. Initial deployment and execution of test scenarios were supported by MICCoM, as part of the Computational Materials Sciences Program funded by the U.S. Department of Energy, Office of Science, Basic Energy Sciences, Materials Sciences and Engineering Division, under grant DE-AC02-06CH11357

REFERENCES

- [1] 3T Software Labs GmbH. Mongochef, 2016. accessed on 2016/05/16.
- [2] Mark James Abraham, Teemu Murtola, Roland Schulz, Szilárd Páll, Jeremy C. Smith, Berk Hess, and Erik Lindahl. GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. *SoftwareX*, 1-2:19–25, 2015.
- [3] Frank H. Allen. The Cambridge Structural Database: a quarter of a million crystal structures and rising. *Acta Crystallogr. Sect. B*, 58(3):380–388, 2002.
- [4] Joshua A Anderson and Sharon C Glotzer. The development and expansion of hoomd-blue through six years of gpu proliferation. *arXiv preprint arXiv:1308.5587*, pages 1–13, 2013.

- [5] Joshua A Anderson, M. Eric Irrgang, and Sharon C Glotzer. Scalable Metropolis Monte Carlo for simulation of hard shapes. *Comput. Phys. Commun.*, 204:21–30, 2016.
- [6] Joshua A. Anderson, Eric Jankowski, Thomas L. Grubb, Michael Engel, and Sharon C. Glotzer. Massively parallel Monte Carlo for many-particle simulations on GPUs. *J. Comput. Phys.*, 254:27–38, 2013.
- [7] H M Berman, John Westbrook, Zukang Feng, Gary Gilliland, T N Bhat, Helge Weissig, Ilya N Shindyalov, and Philip E Bourne. The Protein Data Bank. *Nucleic Acids Res.*, 28(1):235–242, 2000.
- [8] P. F. Damasceno, M. Engel, and Sharon C. Glotzer. Predictive Self-Assembly of Polyhedra into Complex Structures. *Science*, 337(6093):453–457, 2012.
- [9] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *2009 IEEE Conf. Comput. Vis. Pattern Recognit.*, pages 248–255. IEEE, 2009.
- [10] Christopher C. Fischer, Kevin J. Tibbetts, Dane Morgan, and Gerbrand Ceder. Predicting crystal structure by merging data mining with quantum mechanics. *Nat. Mater.*, 5(8):641–646, 2006.
- [11] Ian Foster. Globus online: Accelerating and democratizing science through cloud-based services. *IEEE Internet Comput.*, 15(3):70–73, 2011.
- [12] G. Brandl and the Sphinx team. Sphinx Documentation, 2016. accessed on 2016/05/16.
- [13] Colin R. Groom and Frank H. Allen. The Cambridge Structural Database in Retrospect and Prospect. *Angew. Chemie Int. Ed.*, 53(3):662–671, 2014.
- [14] Anubhav Jain, Shyue Ping Ong, Geoffroy Hautier, Wei Chen, William Davidson Richards, Stephen Dacek, Shreyas Cholia, Dan Gunter, David Skinner, Gerbrand Ceder, and Kristin A. Persson. Commentary: The Materials Project: A materials genome approach to accelerating materials innovation. *APL Mater.*, 1(1):011002, 2013.
- [15] Anand Kumar, Vladimir Grupcev, Meryem Berrada, Joseph C Fogarty, Yi-Cheng Tu, Xingquan Zhu, Sagar A Pandit, and Yuni Xia. DCMS: A data analytics and management system for molecular simulation. *J. Big Data*, 2(1):1–22, 2014.
- [16] Robert Martin. The Clean Architecture, 2012. accessed on 2016/05/16.
- [17] MongoDB, Inc. MongoDB, 2016. accessed on 2016/06/06.
- [18] Oracle Corporation. MySQL, 2016. accessed on 2016/06/06.
- [19] Paralect, Inc. Robomongo, 2016. accessed on 2016/05/16.
- [20] Giovanni Pizzi, Andrea Cepellotti, Riccardo Sabatini, Nicola Marzari, and Boris Kozinsky. AiiDA: automated interactive infrastructure and database for computational science. *Comput. Mater. Sci.*, 111:218–230, 2016.
- [21] Steve Plimpton. Fast Parallel Algorithms for Short-Range Molecular Dynamics. *J. Comput. Phys.*, 117(1):1–19, 1995.
- [22] Michael Shirts and Vijay S. Pande. Screen Savers of the World Unite! *Science*, 290(5498):1903–1904, 2000.
- [23] The iRODS Consortium. Integrated Rule-Oriented System (iRODS), 2016. accessed on 2016/06/06.
- [24] Monika Thol, Gabor Rutkai, Andreas Köster, Rolf Lustig, Roland Span, and Jadran Vrabec. Equation of State for the Lennard-Jones Fluid. *J. Phys. Chem. Ref. Data*, 45(2):023101–36, 2016.
- [25] John Towns, Timothy Cockerill, Maytal Dahan, Ian Foster, Kelly Gaither, Andrew Grimshaw, Victor Hazlewood, Scott Lathrop, Dave Lifka, Gregory D Peterson, Ralph Roskies, J Ray Scott, and Nancy Wilkins-Diehr. XSEDE: Accelerating Scientific Discovery. *Comput. Sci. Eng.*, 16(5):62–74, 2014.